

FIG. 1

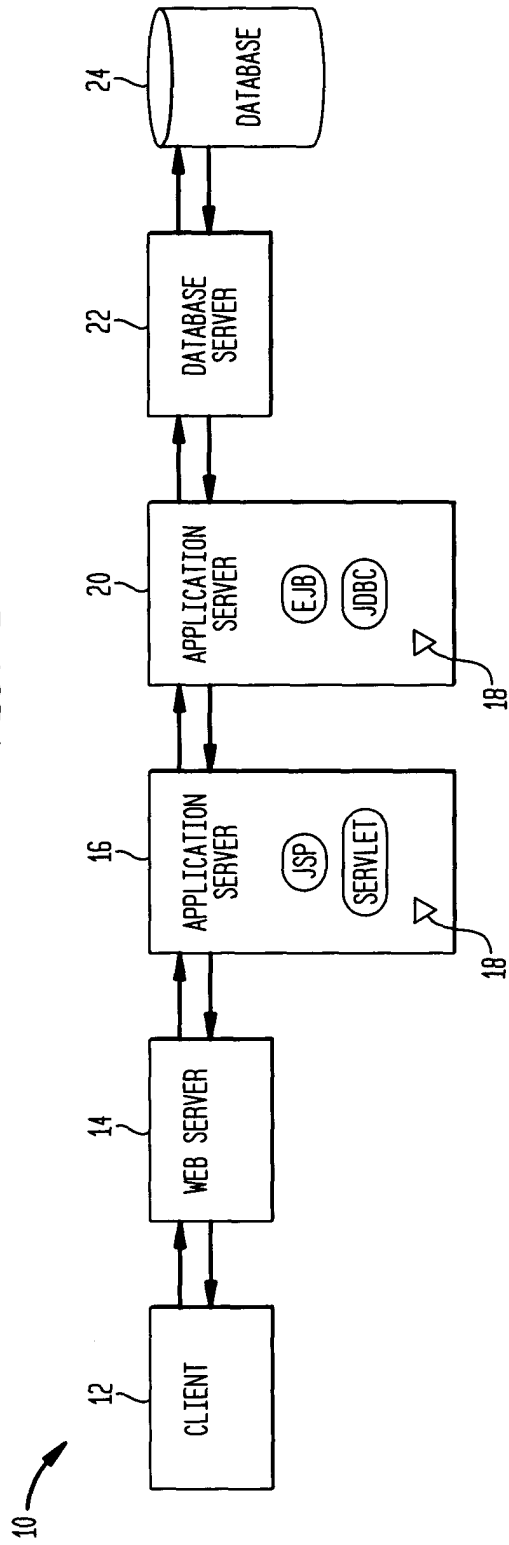
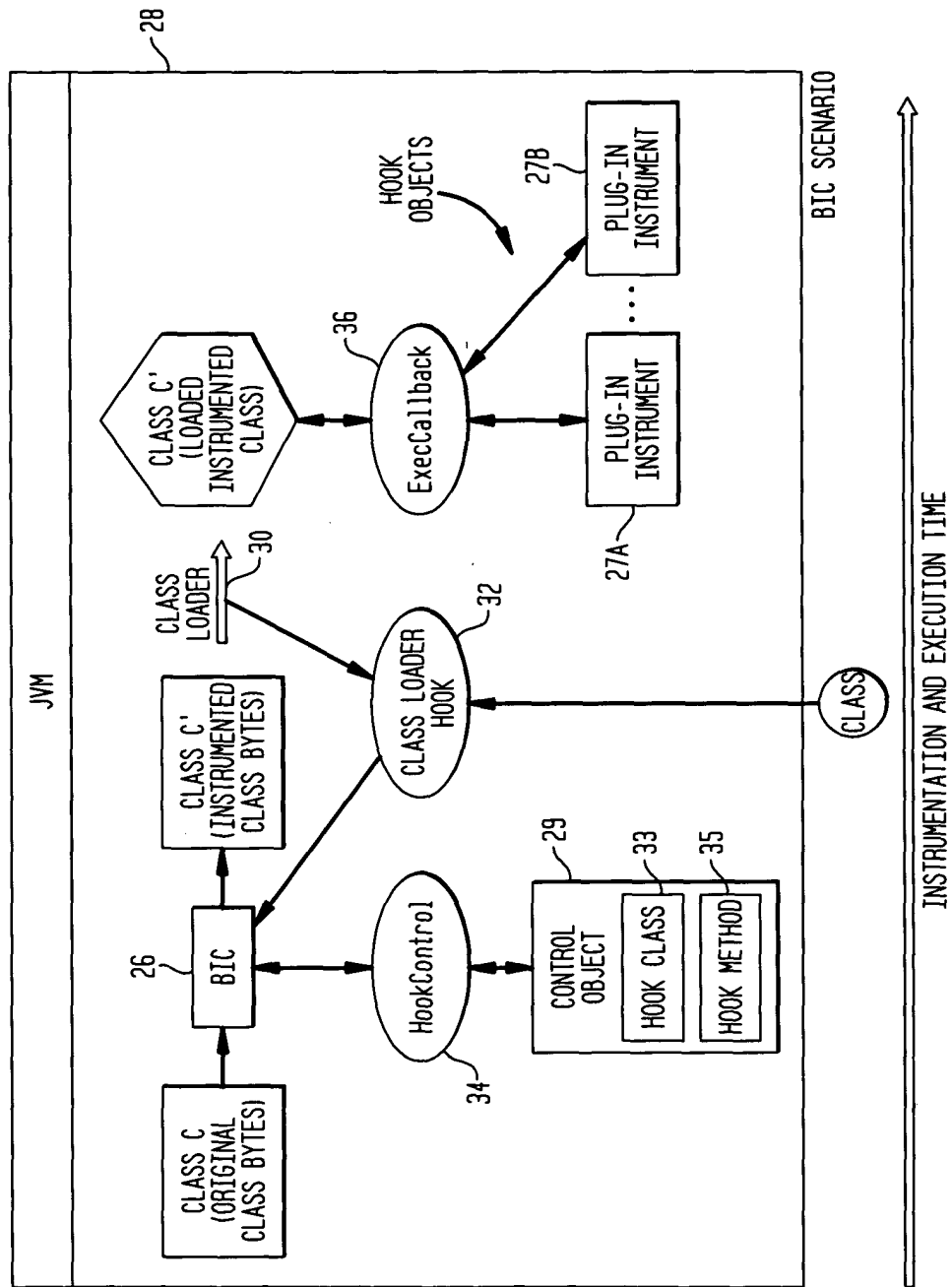


FIG. 2



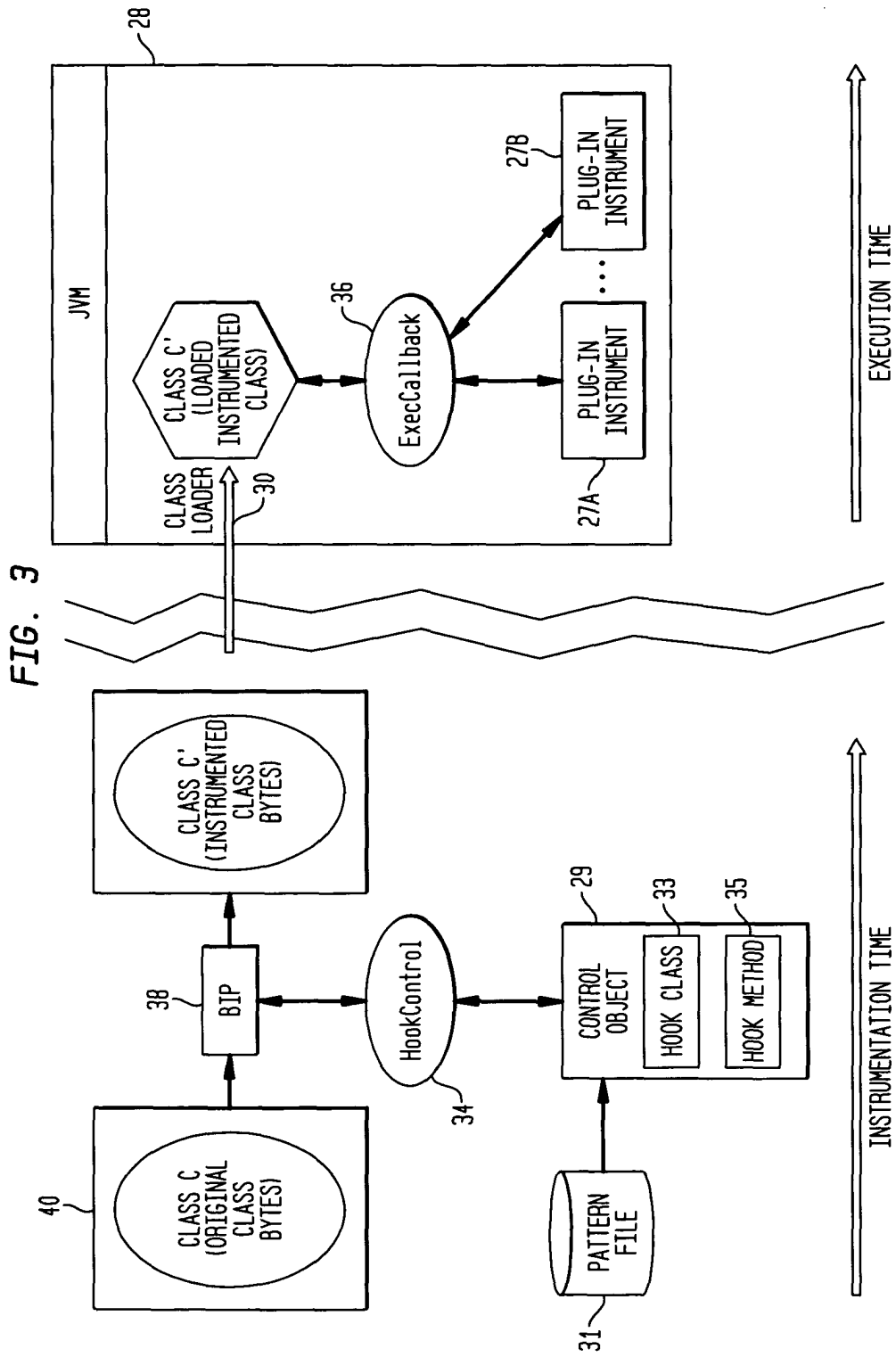


FIG. 4

```

public java.lang.Object hookClass (
    java.lang.String classname, 402
    java.lang.String [] methods, 404
    java.lang.String [] superclasses, 406
    java.lang.String [] superinterfaces, 408
    java.lang.StringBuffer getHookArg) 410
    } 400

```

FIG. 5

```

public int hookMethod(
    java.lang.Object classcontext, 502
    java.lang.String classname, 504
    java.lang.String methodname, 506
    java.lang.String [] superinterfaces, 508
    java.lang.StringBuffer defMethodArg) 510
    } 500

public static final int DO_NOT_HOOK; 522
public static final int HOOK_NO_ARGS; 524
public static final int HOOK_WITH_ARGS; 526
public static final int HOOK_WITH_ARG1; 528
public static final int HOOK_WITH_ARG1_2; 530
public static final int HOOK_WITH_ARG2; 532
    } 520

```

FIG. 6A

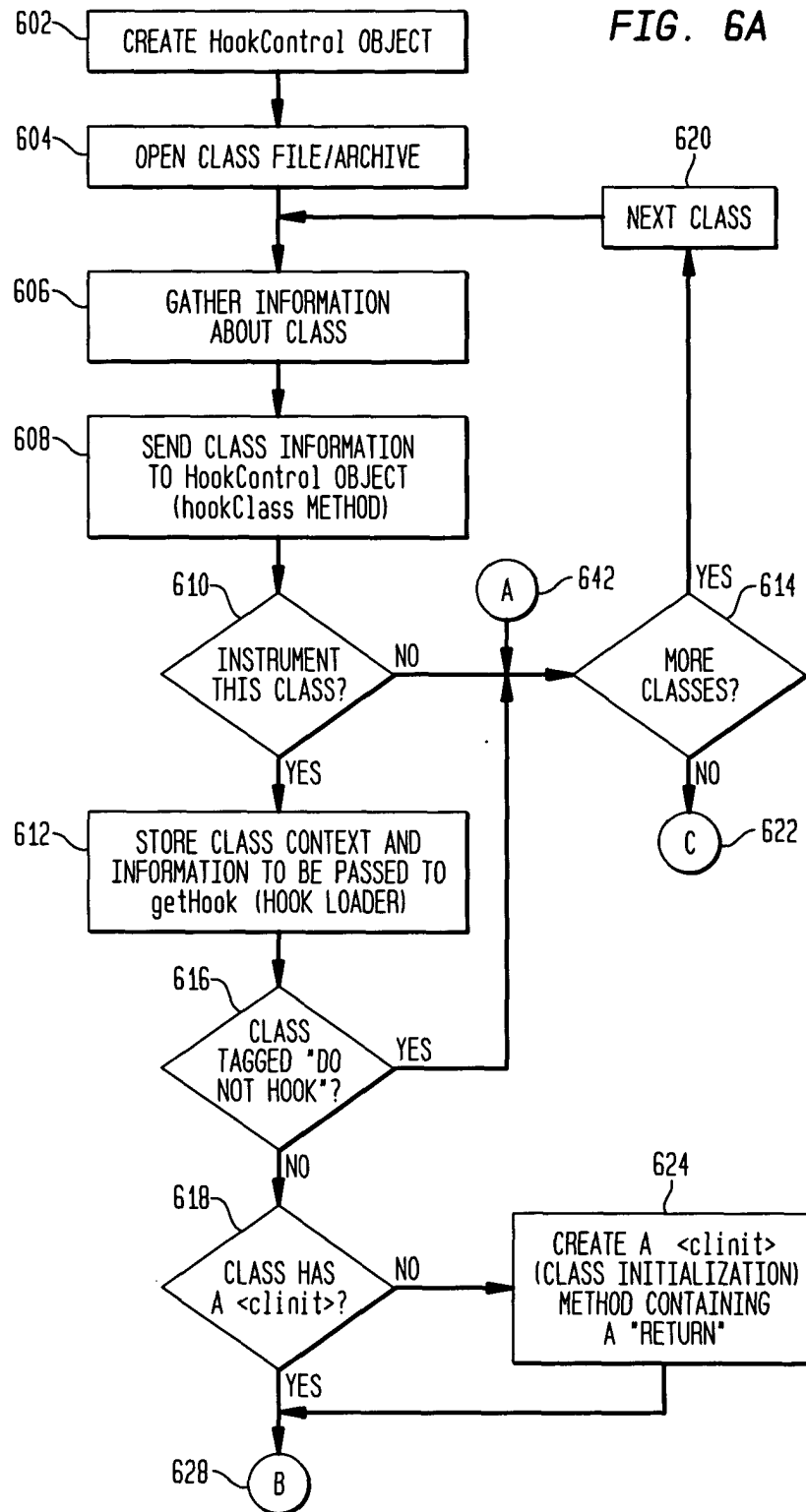


FIG. 6B

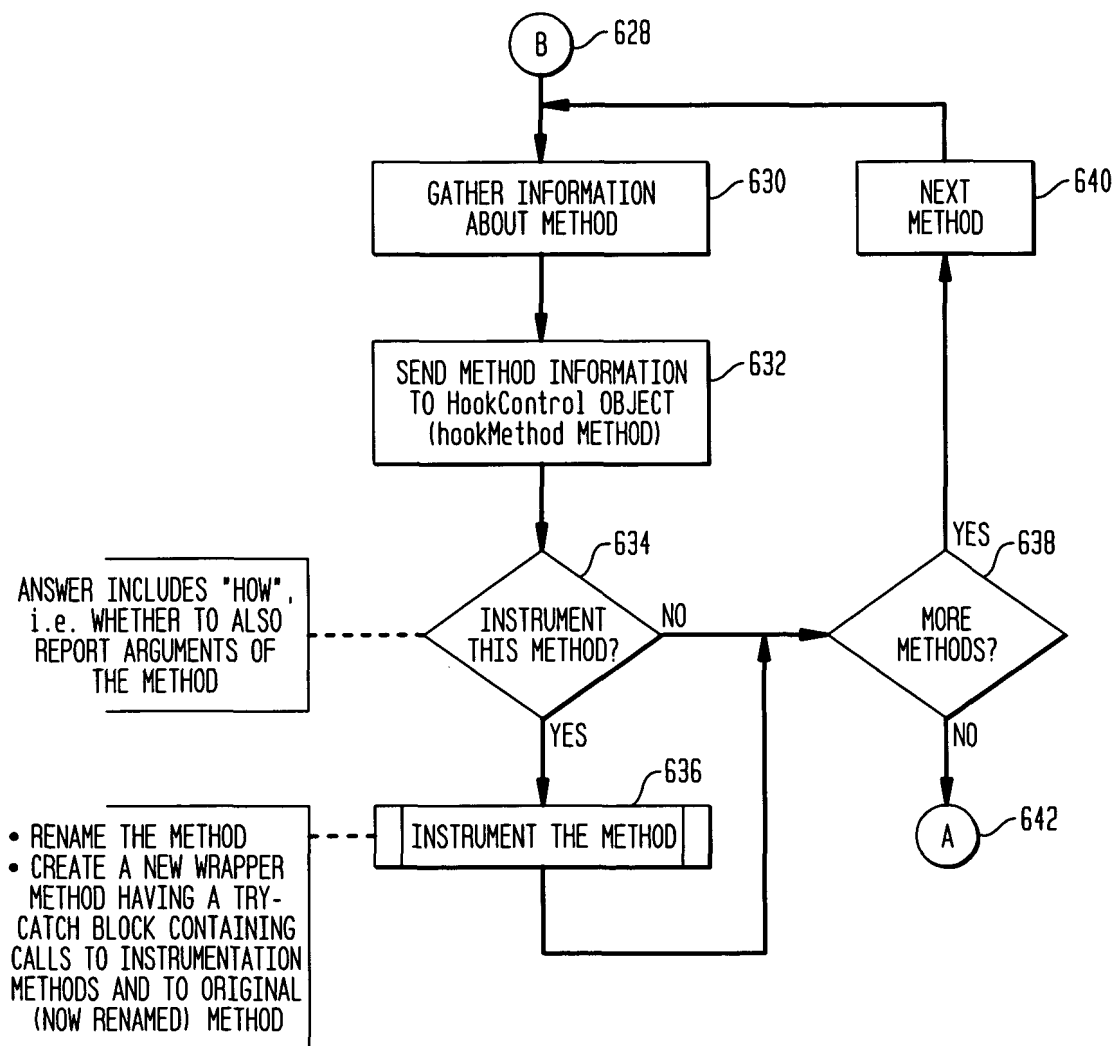


FIG. 6C

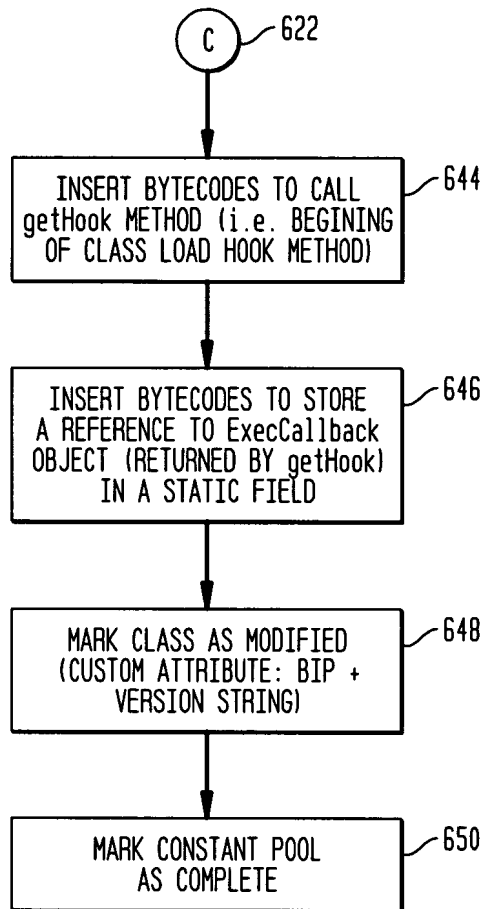


FIG. 7

```

public TradeResult buy(String string, int i)
{
    Object object; 728
    Throwable throwable;
    TradeResult tradeResult; 718
734 if ($BIP$hook = null)
    {
        $BIP$installHook(); 702
726 object=$BIP$hook.methodEntry($BIP$ref_C,$BIP$ref_M0,this,2);
730 if (object!=null) 708
    {
        $BIP$hook.reportArg(object,$BIP$ref_C,$BIP$ref_M0,1,string);
        $BIP$hook.reportArg(object,$BIP$ref_C,$BIP$ref_M0,2,i);
    }
714 {
    try 720 710 704
    {
        tradeResult = $BIP$buy(string, i);
    }
    catch (Throwable throwable) 716 712
    {
        $BIP$hook.methodException(object,$BIP$ref_C,$BIP$ref_M0,throwable);
        throw throwable;
    }
732 if (object!=null) 706
    {
        $BIP$hook.methodExit(object,$BIP$ref_C,$BIP$ref_M0,tradeResult);
        return tradeResult;
    }
    ... 724 722
private TradeResult $BIP$buy(String string,int i)
    ... Original, unmodified conents of buy
}

```

700

701

FIG. 8A

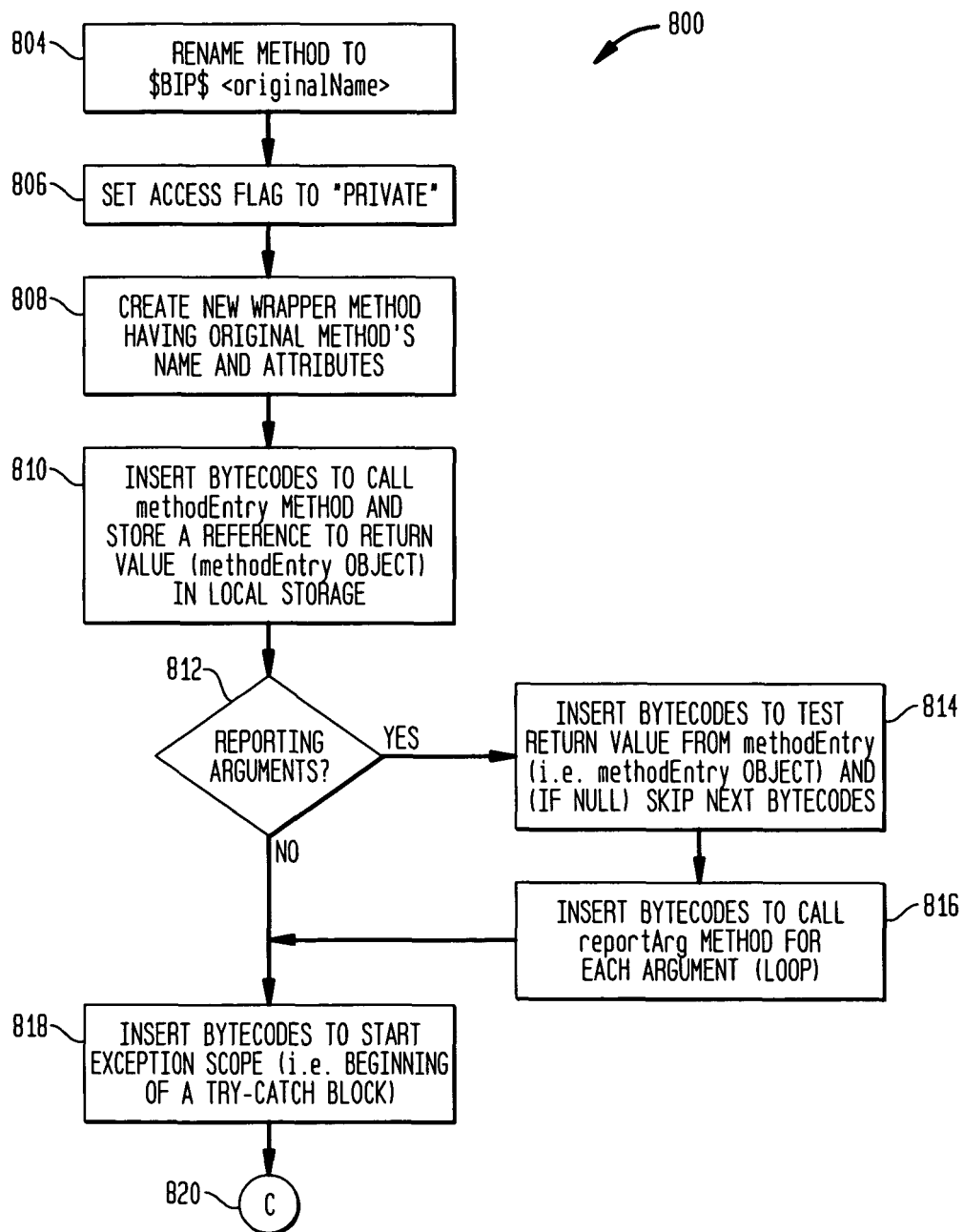


FIG. 8B

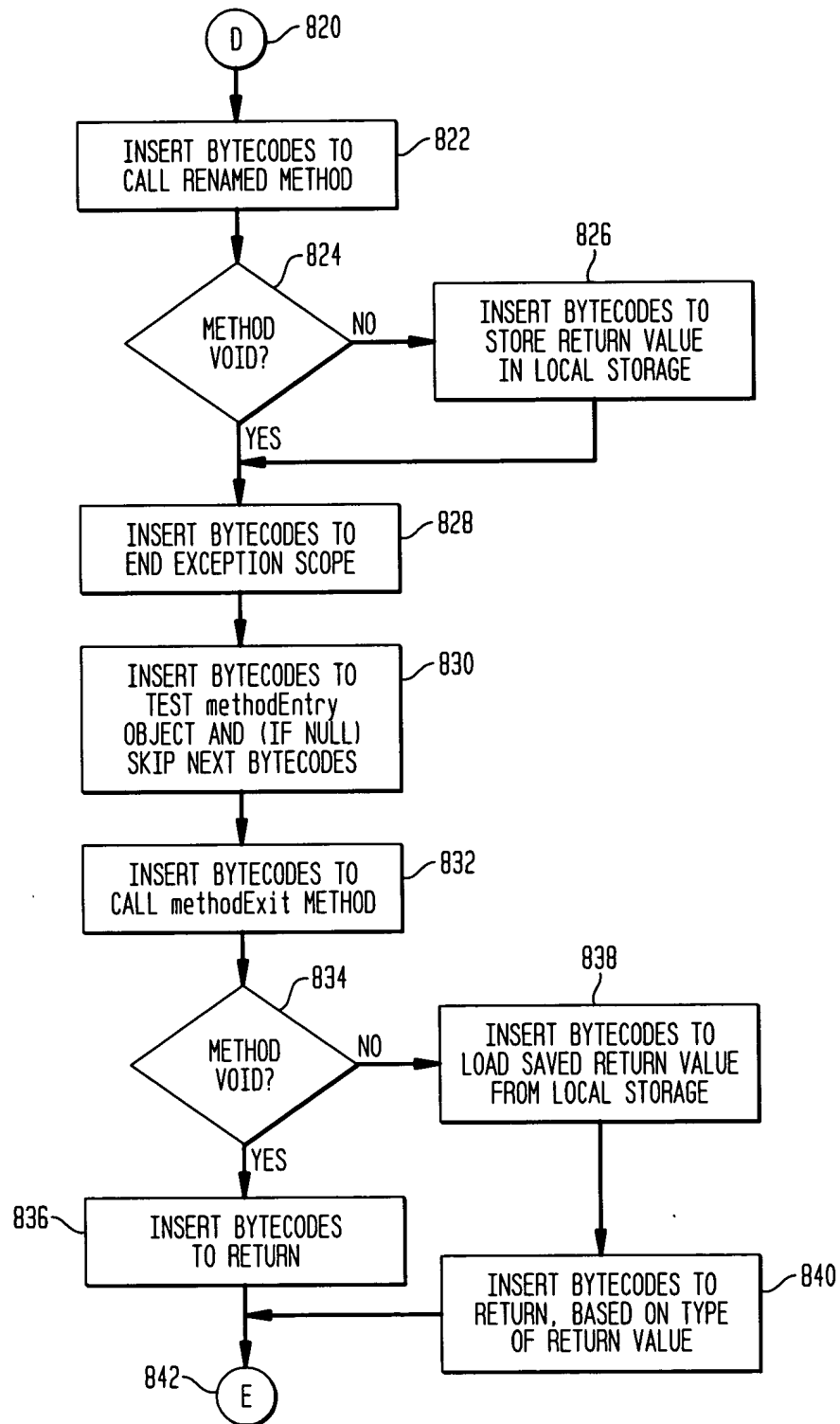


FIG. 8C

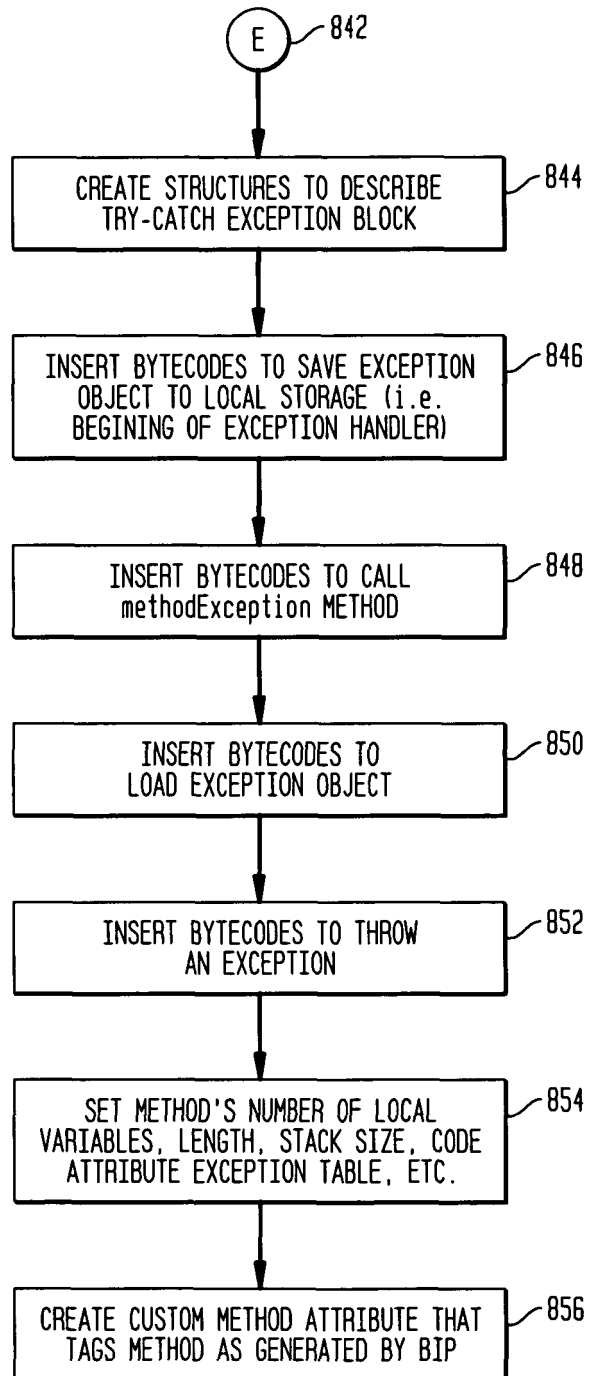


FIG. 9

```

public java.lang.Object classLoadStart (
    java.lang.String  classname, 902
    java.lang.class   classObj, 904
    int methods) 906
    } 900

public java.lang.Object defMethod (
    java.lang.Object  classref, 922
    java.lang.String  methodname, 924
    java.lang.String  methodkind) 926
    } 920

public void classLoadEnd(
    java.lang.Object  classref) 942
    } 940

```

FIG. 10

```

public java.lang.Object methodEntry (
    java.lang.Object  classref, 1002
    java.lang.Object  methodref, 1004
    java.lang.Object  instance, 1006
    int args) 1008
    } 1000

public void reportArg (
    java.lang.Object  context, 1022
    java.lang.Object  classref, 1024
    java.lang.Object  methodref, 1026
    int argNumber, 1028
    java.lang.Object  methodArg) 1030
    } 1020

public void methodExit (
    java.lang.Object  context, 1042
    java.lang.Object  classref, 1044
    java.lang.Object  methodref, 1046
    java.lang.Object  result) 1048
    } 1040

```

FIG. 11

```

public java.lang.Object methodEntryOneArg(
    java.lang.Object classref,
    java.lang.Object methodref,
    java.lang.Object instance,
    java.lang.Object selectedArg) 1102
    } 1100

public void methodException (
    java.lang.Object context,
    java.lang.Object classref,
    java.lang.Object methodref,
    java.lang.Throwable e) 1122
    } 1120

```

FIG. 12

```

public static ExecCallback getHook (
    java.lang.String className, 1202
    java.lang.String classKind, 1204
    java.lang.String className, 1206
    java.lang.String classVersion, 1208
    java.lang.String interface Version) 1210
    } 1200

```

FIG. 13A

1300

```

// $Source: /data1/nebula/ccm/jade/ccm/import/arra_jlink/i2/bip/hook/RCS/NullExec?Callback.java,v $
// $Revision: 1.8 $ $Date: 2001/08/28 14:56:29 $ $Author: arav $
package i2.bip.hook;
/** An implementation of the ExecCallback that does nothing.
 * A suitable base class for a custom hook class.
 */
public class NullExecCallback
    // Explicit DoNotHook for BIC testing
    implements ExecCallback, DoNotHook {

    // Called at start of class initialization
    // Returns opaque class ref
    public Object classLoadStart(String classname, Class classObj, int methods)
    {
        return null;
    }

    // Called once for each instrumented method in the class.
    // Returns opaque method ref
    public Object defMethod(
        Object classref,
        String methodname,
        String methodkind)
    {
        return null;
    }

    // End of class initialization instrumentation
    public void classLoadEnd(Object classref) { }


    // Called at instrumented method entry.
    public Object methodEntry(
        Object classref,
        Object methodref,
        Object instance,
        int args)
    {
        return null;    // Disables methodExit & reportArg instrumentation
    }

    // Called at instrumented method entry when single arg requested.

```

FIG. 13B

1300



```

public Object methodEntryOneArg(
    Object classref,
    Object methodref,
    Object instance,
    Object selectedArg)
{
    return null;    // Disables methodExit & reportArg instrumentation
}

public Object methodEntryOneTwoArg(
    Object classref,
    Object methodref,
    Object instance,
    Object arg1,
    Object arg2)
{
    return null;    // Disables methodExit & reportArg instrumentation
}

// Called at normal instrumented method exit.
// unless returned methodEntry context is null.
public void methodExit(
    Object context,
    Object classref,
    Object methodref,
    Object result) { }

// Overloaded versions of methodExit for primitive return types.
public void methodExit(
    Object context,
    Object classref,
    Object methodref,
    int result) { }    // Covers boolean, byte, char, short, and int
public void methodExit(
    Object context,
    Object classref,
    Object methodref,
    float result) { }
public void methodExit(
    Object context,
    Object classref,
    Object methodref,

```

FIG. 13C

1300

```

    long result) { }
    public void methodExit(
        Object context,
        Object classref,
        Object methodref,
        double result) { }
    public void methodExit(
        Object context,
        Object classref,
        Object methodref) { }

    // Called unconditionally at method exception
    public void methodException(
        Object context,
        Object classref,
        Object methodref,
        Throwable e) { }

    //-----
    // Argument reporting
    //-----

    // Called after instrumented method entry, once per arg. if
    // argument reporting was instrumented.
    public void reportArg(
        Object context,
        Object classref,
        Object methodref,
        int argNumber,           // starts at 1
        Object methodArg)       // The actual argument (reference types)
    {
    }

    // Overloaded versions of reportArg for primitive types.
    public void reportArg(
        Object context,
        Object classref,
        Object methodref,
        int argNumber,           // starts at 1
        int methodArg)          // Covers boolean, byte, char, short, and int
    {
    }

```


FIG. 13D

1300

```

public void reportArg(
    Object context,
    Object classref,
    Object methodref,
    int argNumber,           // starts at 1
    float methodArg)
{
}
public void reportArg(
    Object context,
    Object classref,
    Object methodref,
    int argNumber,           // starts at 1
    long methodArg)
{
}
public void reportArg(
    Object context,
    Object classref,
    Object methodref,
    int argNumber,           // starts at 1
    double methodArg)
{
}
} // class NullExecCallback

```

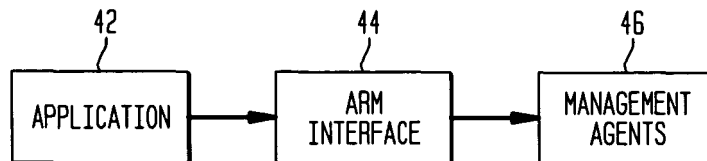
FIG. 14

FIG. 15

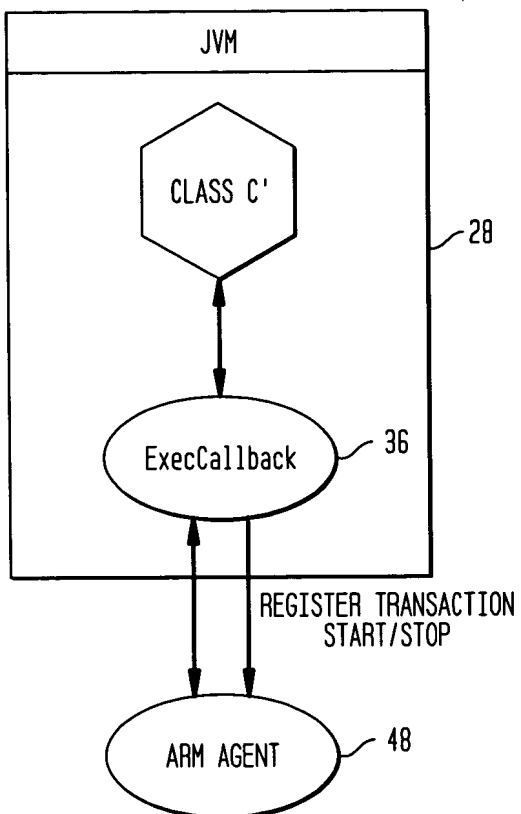


FIG. 16

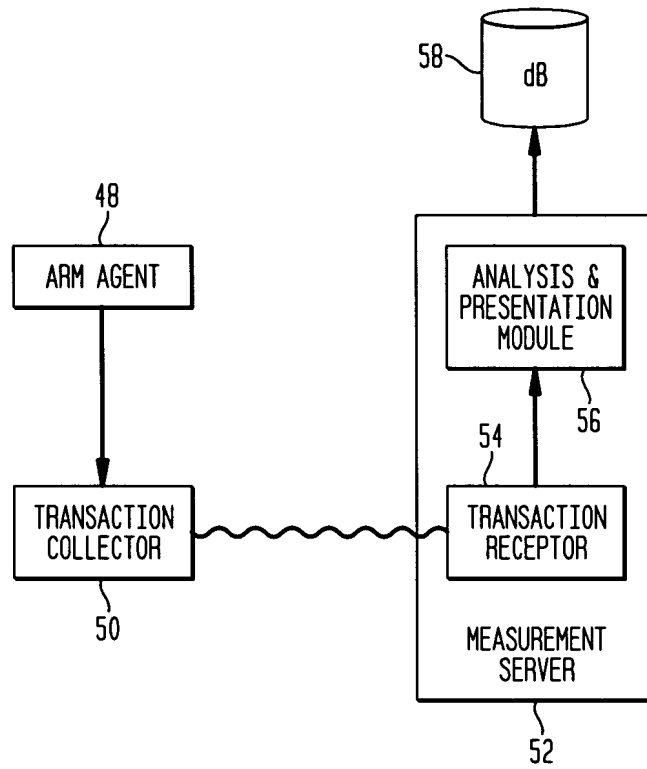


FIG. 17

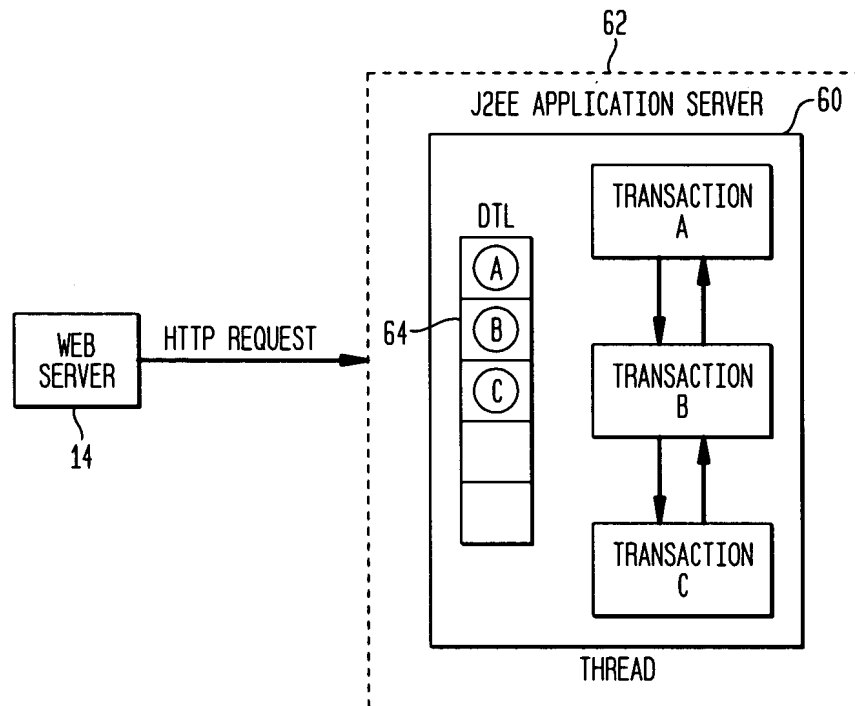


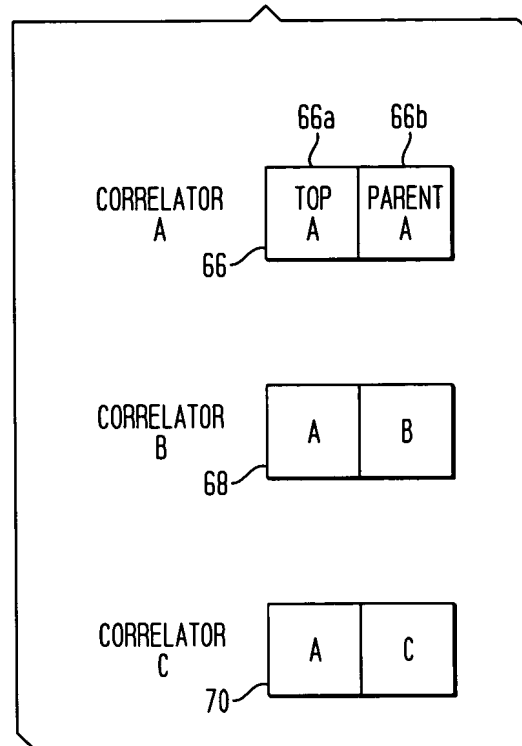
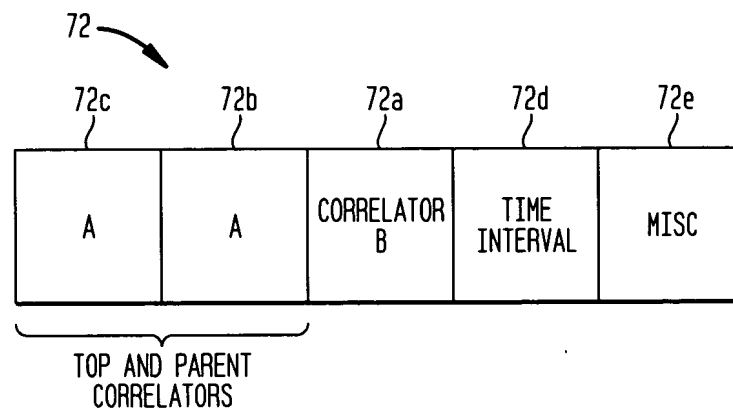
FIG. 18**FIG. 19**

FIG. 20

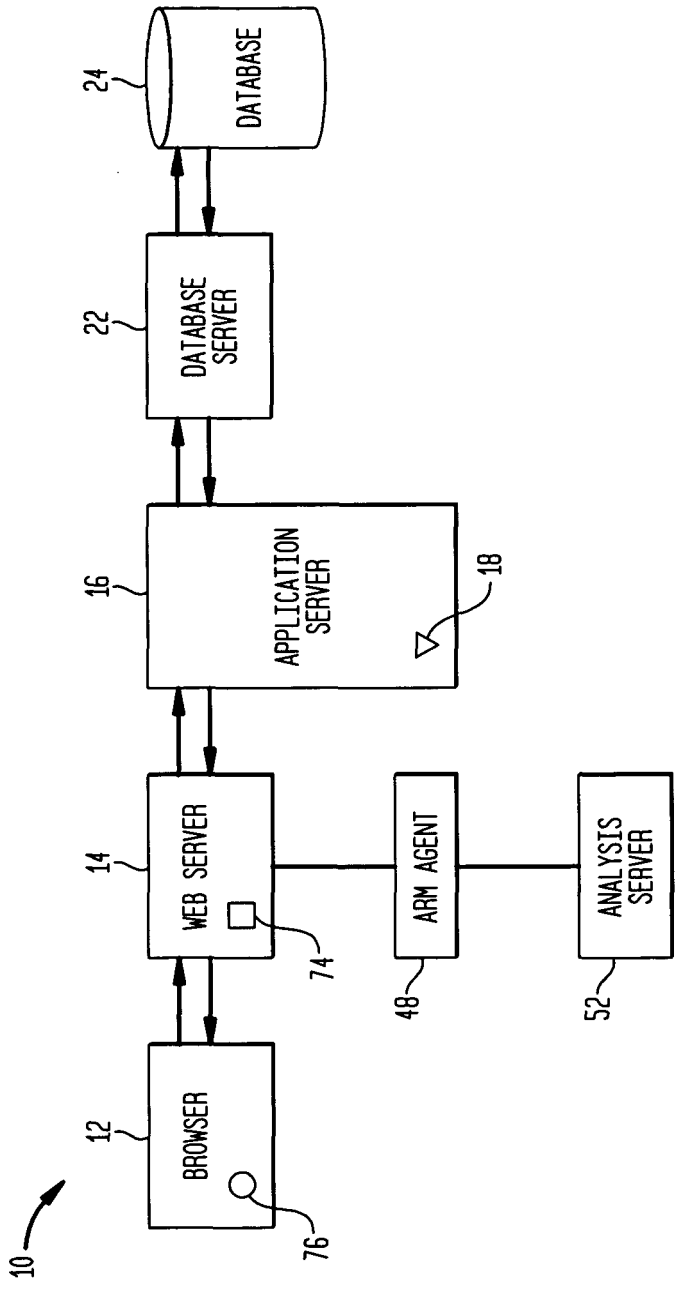


FIG. 21

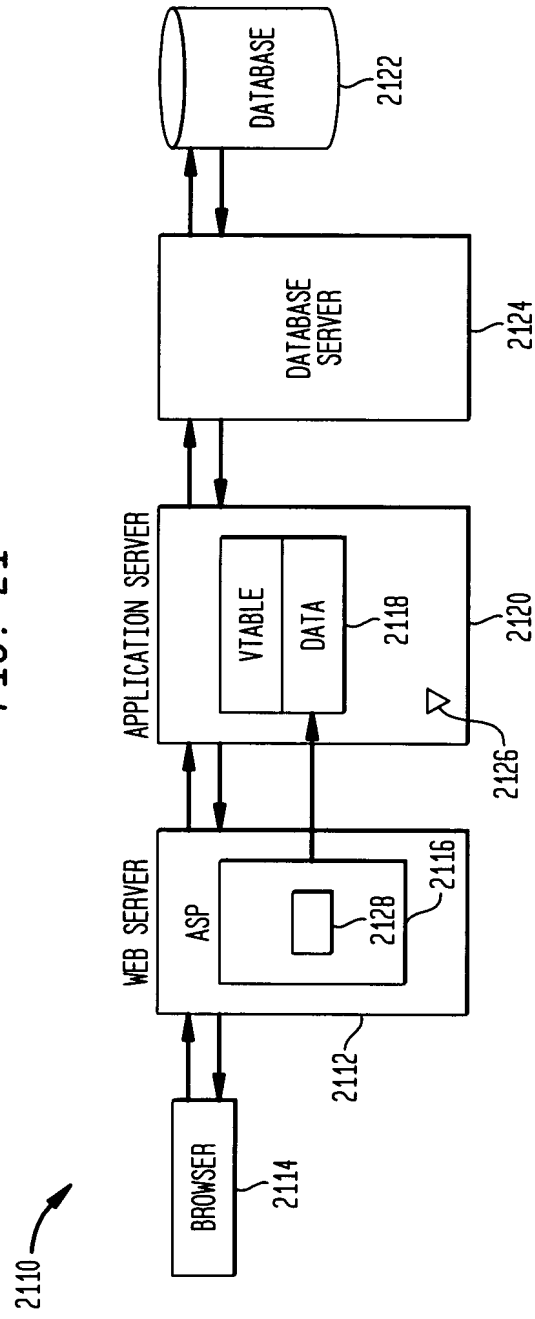


FIG. 22

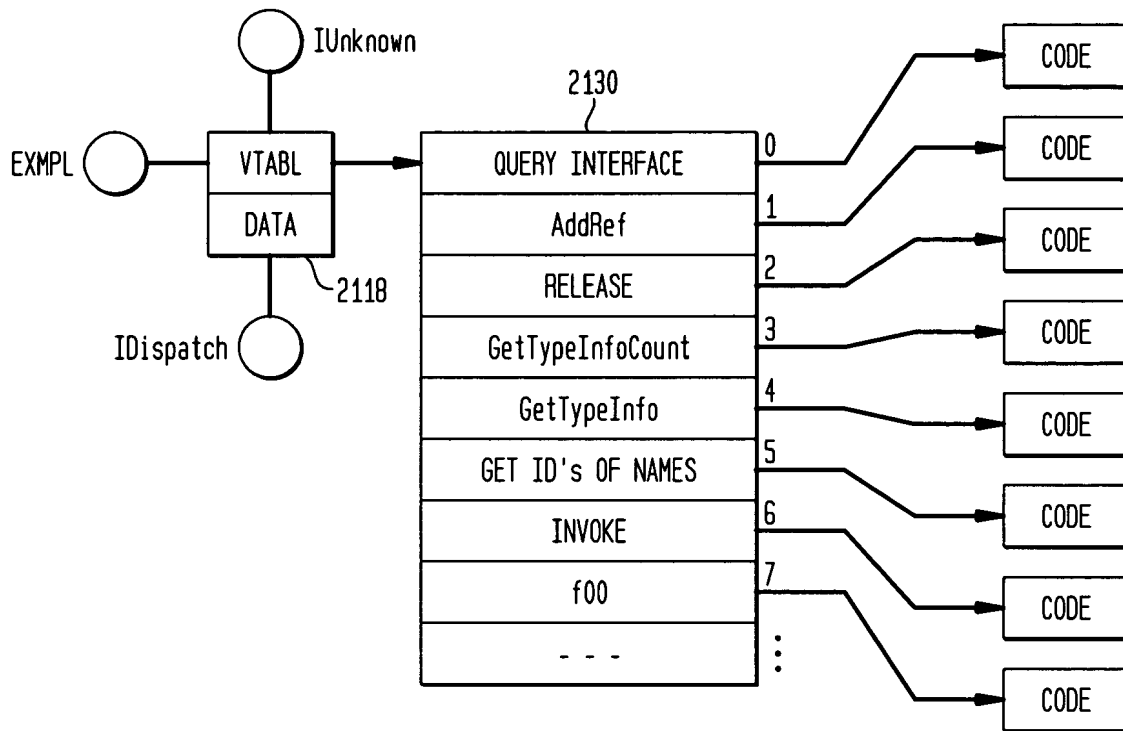


FIG. 23

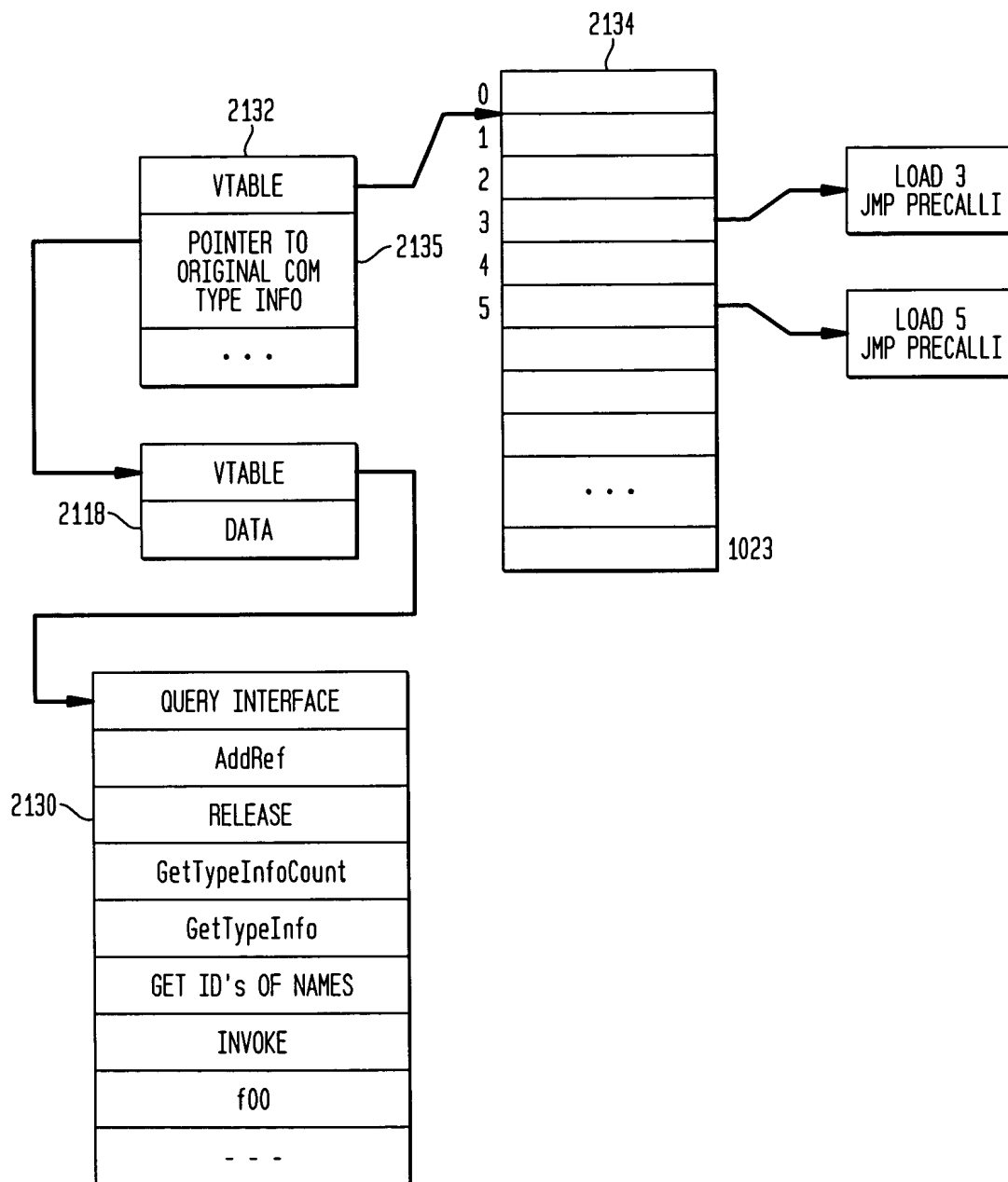


FIG. 24

```

PrecallInterceptor UNIVERSAL COM METHOD (METHOD #){
    DETERMINE ARGUMENTS NEEDED FOR METHOD #
    ARM START
    CALL ORIGINAL METHOD #
    ARM STOP
}

```

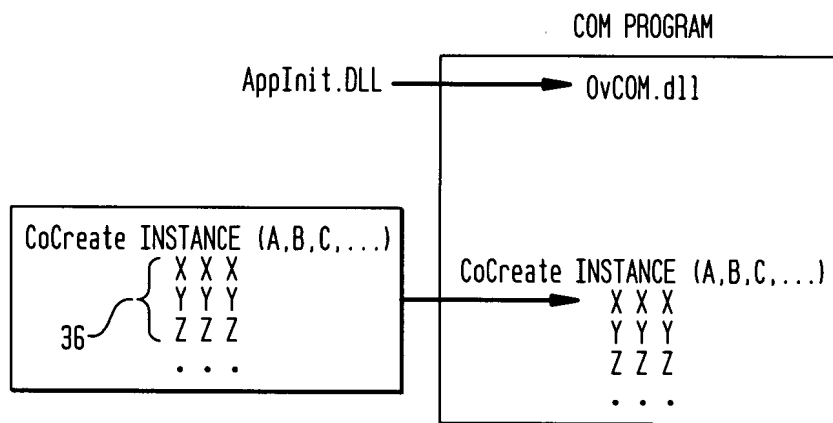
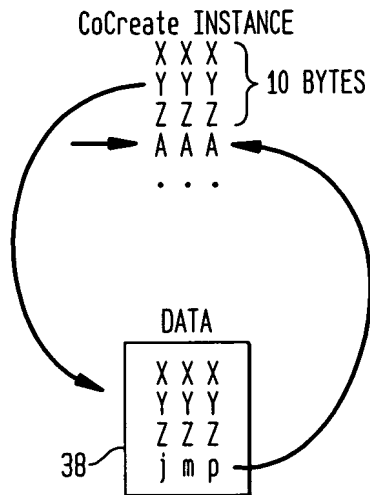
FIG. 25

FIG. 26**FIG. 27**

```

OVTA CoCreateInstance (A,B,C) {
    :
    :
    CALL CoCreateInstance (A,B,C) {
        :
        :
        ACCESS B
        WRAP OBJECT REFERRED BY B
        SET B TO POINT TO WRAPPER OBJECT
        RETURN TO ORIGINAL CALLER
    }
}

```

FIG. 28

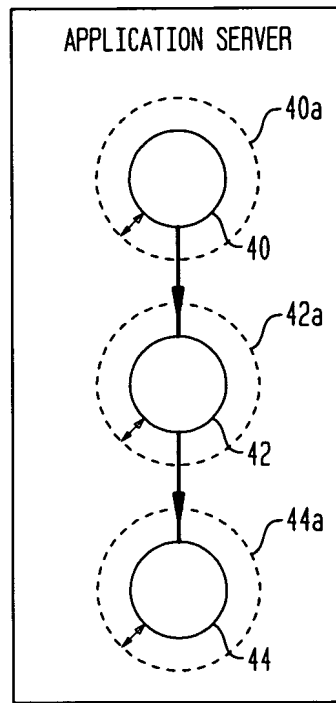


FIG. 29

